Implementation of a simple ANN with numpy

Prosad Kumar Das

September 26, 2024

1 Import Libraries

[152]:
```
import sys
print("Python3 version", sys.version)
import numpy as np
print("Numpy version: ", np.__version__)
import pandas as pd
print("Pandas version: ", pd.__version__)
import matplotlib.pyplot as plt
%matplotlib inline
```

```
Python3 version 3.8.19 | packaged by conda-forge | (default, Mar 20 2024,
12:47:35)
[GCC 12.3.0]
Numpy version:  1.24.4
Pandas version:  2.0.3
```

2 Load Data and QC

[153]:
```
# Set working directory
import os
os.chdir("/home/prasad/mnist_numpy/")
path = os.listdir()
print(path)
```

```
['train.csv', 'train.csv.zip', 'test.csv', 'test.csv.zip',
'sample_submission.csv']
```

[154]:
```
# Load data ito dataframes
train_data = pd.read_csv("train.csv")
test_data = pd.read_csv("test.csv")
```

[155]:
```
# Check train file
train_data.head()
```

[155]:
```
   label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0      1       0       0       0       0       0       0       0       0
1      0       0       0       0       0       0       0       0       0
2      1       0       0       0       0       0       0       0       0
3      4       0       0       0       0       0       0       0       0
4      0       0       0       0       0       0       0       0       0

   pixel8  ...  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  ...         0         0         0         0         0         0
1       0  ...         0         0         0         0         0         0
2       0  ...         0         0         0         0         0         0
3       0  ...         0         0         0         0         0         0
4       0  ...         0         0         0         0         0         0
```

```
   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 785 columns]
```

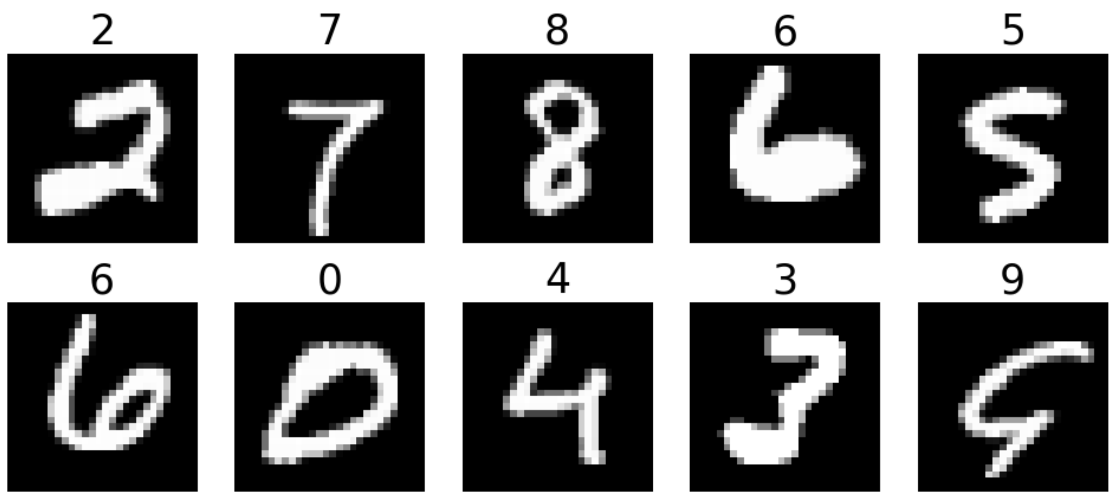
[156]:
```python
# Check test file
test_data.head()
```

[156]:
```
   pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
0       0       0       0       0       0       0       0       0       0
1       0       0       0       0       0       0       0       0       0
2       0       0       0       0       0       0       0       0       0
3       0       0       0       0       0       0       0       0       0
4       0       0       0       0       0       0       0       0       0

   pixel9  ...  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  ...         0         0         0         0         0         0
1       0  ...         0         0         0         0         0         0
2       0  ...         0         0         0         0         0         0
3       0  ...         0         0         0         0         0         0
4       0  ...         0         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 784 columns]
```

[157]:
```python
# Set up the data

# labels for ground check
y_train = train_data["label"].values

# Input vectors
X_train = train_data.drop(columns = ["label"]).values/255

# Test data
X_test = test_data.values/255
```

[158]:
```python
# Check input images by plotting pixels
fig, axes = plt.subplots(2,5, figsize=(12,5))
axes = axes.flatten()
idx = np.random.randint(0,42000,size=10)
for i in range(10):
    axes[i].imshow(X_train[idx[i],:].reshape(28,28), cmap='gray')
    axes[i].axis('off') # hide the axes ticks
```

```python
    axes[i].set_title(str(int(y_train[idx[i]])), color= 'black', fontsize=25)
plt.show()
```



3 Function Definitions

[159]:

```python
# ReLU activation function
def relu(x: np.array) -> np.array:
    """
    Vectorized implementation of ReLU
    -input: A 1-dim numpy array
    -output: Returns A 1-dim array transformed with ReLU
    """

    x[x<0] = 0
    return x
```

[160]:

```python
# Hypothesis function
def h(X: np.array, W: np.array, b: np.array) -> np.array:
    """
    Hypothesis function: simple FNN with 1 hidden layer
    Layer 1: Input
    Layer 2: Hidden layer, dimension implied by the arguments W[0],b
    Layer 3: Output layer, dimension implied by the arguments W[1]
    """

    # Layer 1 is input layer
    a1 = X

    # Layer 1 (input layer) -> Layer 2 (hidden layer)
    z1 = np.matmul(X, W[0]) + b[0]

    # Layer 2 activation
    a2 = relu(z1)
```

```
    # Layer 2 (hidden layer) -> Layer 3 (output layer)
    z2 = np.matmul(a2, W[1])

    # Apply SoftMax on z2
    s = np.exp(z2)
    total = np.sum(s, axis = 1).reshape(-1, 1)
    sigma = s/total
    return sigma
```

[161]:
```
# SoftMax function
def softmax(X_in: np.array, weights: np.array) -> np.array:
    """
    Activation function for the last FC layer: softmax function
    -output: K probabilities represent an estimate of P(y=k|X_in; weights) for k=1,...
 ↪,K
    the weights has shape (n, K)
    n: the number of features X_in has
    n = X_in.shape[1]
    K: the number of classes
    K = 10
    """

    s = np.exp(np.matmul(X_in,weights))
    total = np.sum(s, axis=1).reshape(-1,1)
    return s/total
```

[162]:
```
# Loss function
def loss(y_pred: np.int64, y_true: np.int64):
    """
    Loss function: cross entropy with an L^2 regularization
    y_true: ground truth, of shape (N, )
    y_pred: prediction made by the model, of shape (N, K)
    N: number of samples in the batch
    K: global variable, number of classes
    """

    global K
    K = 10
    N = len(y_true)
    # loss_sample stores the cross entropy for each sample in X
    # convert y_true from labels to one-hot-vector encoding
    y_true_one_hot_vec = (y_true[:,np.newaxis] == np.arange(K))
    loss_sample = (np.log(y_pred) * y_true_one_hot_vec).sum(axis=1)
    # loss_sample is a dimension (N,) array
    # for the final loss, we need take the average
    return -np.mean(loss_sample)
```

[163]:
```
# Backpropagation
def backprop(W, b, X, y, alpha = 1e-4):
    """
    Step 1: explicit forward pass h(X;W,b)
    Step 2: backpropagation for dW and db
```

```
    """
    K = 10
    N = X.shape[0]

    ### Step 1:
    # layer 1 = input layer
    a1 = X
    # layer 1 (input layer) -> layer 2 (hidden layer)
    z1 = np.matmul(X, W[0]) + b[0]
    # layer 2 activation
    a2 = relu(z1)

    # one more layer

    # layer 2 (hidden layer) -> layer 3 (output layer)
    z2 = np.matmul(a2, W[1])
    s = np.exp(z2)
    total = np.sum(s, axis=1).reshape(-1,1)
    sigma = s/total

    ### Step 2:

    # layer 2->layer 3 weights' derivative
    # delta2 is \partial L/partial z2, of shape (N,K)
    y_one_hot_vec = (y[:,np.newaxis] == np.arange(K))
    delta2 = (sigma - y_one_hot_vec)
    grad_W1 = np.matmul(a2.T, delta2)

    # layer 1->layer 2 weights' derivative
    # delta1 is \partial a2/partial z1
    # layer 2 activation's (weak) derivative is 1*(z1>0)
    delta1 = np.matmul(delta2, W[1].T)*(z1>0)
    grad_W0 = np.matmul(X.T, delta1)

    # Possible student project: extra layer of derivative

    # no derivative for layer 1

    # the alpha part is the derivative for the regularization
    # regularization = 0.5*alpha*(np.sum(W[1]**2) + np.sum(W[0]**2))


    dW = [grad_W0/N + alpha*W[0], grad_W1/N + alpha*W[1]]
    db = [np.mean(delta1, axis=0)]
    # dW[0] is W[0]'s derivative, and dW[1] is W[1]'s derivative; similar for db
    return dW, db
```

4 Hyper-parameters and network initialization

[164]:
```
eta = 5e-1 # learning rate or, step size
alpha = 1e-6 # regularization
gamma = 0.99 # RMSprop
eps = 1e-3 # RMSprop
num_iter = 1000 # number of iterations of gradient descent
n_H = 256 # number of neurons in the hidden layer
n = X_train.shape[1] # number of pixels in an image
K = 10 # number of output classes
```

[165]:
```
# Initialization
np.random.seed(1127825)
W = [1e-1*np.random.randn(n, n_H), 1e-1*np.random.randn(n_H, K)]
b = [np.random.randn(n_H)]
```

5 Gradient Descent: training of the network

[168]:
```
%%time
gW0 = gW1 = gb0 = 1

for i in range(num_iter):
    dW, db = backprop(W,b,X_train,y_train,alpha)

    gW0 = gamma*gW0 + (1-gamma)*np.sum(dW[0]**2)
    etaW0 = eta/np.sqrt(gW0 + eps)
    W[0] -= etaW0 * dW[0]

    gW1 = gamma*gW1 + (1-gamma)*np.sum(dW[1]**2)
    etaW1 = eta/np.sqrt(gW1 + eps)
    W[1] -= etaW1 * dW[1]

    gb0 = gamma*gb0 + (1-gamma)*np.sum(db[0]**2)
    etab0 = eta/np.sqrt(gb0 + eps)
    b[0] -= etab0 * db[0]

    if i % 500 == 0:
        # sanity check 1
        y_pred = h(X_train,W,b)
        print("Cross-entropy loss after", i+1, "iterations is {:.8}".format(
              loss(y_pred,y_train)))
        print("Training accuracy after", i+1, "iterations is {:.4%}".format(
              np.mean(np.argmax(y_pred, axis=1)== y_train)))

        # sanity check 2
        print("gW0={:.4f} gW1={:.4f} gb0={:.4f}\netaW0={:.4f} etaW1={:.4f} etab0={:.
↪4f}"
              .format(gW0, gW1, gb0, etaW0, etaW1, etab0))

        # sanity check 3
        print("|dW0|={:.5f} |dW1|={:.5f} |db0|={:.5f}"
```

```
            .format(np.linalg.norm(dW[0]), np.linalg.norm(dW[1]), np.linalg.
 ↪norm(db[0])), "\n")

        # reset RMSprop
        gW0 = gW1 = gb0 = 1

y_pred_final = h(X_train,W,b)
print("Final cross-entropy loss is {:.8}".format(loss(y_pred_final,y_train)))
print("Final training accuracy is {:.4%}".format(np.mean(np.argmax(y_pred_final,␣
 ↪axis=1)== y_train)))
```

```
Cross-entropy loss after 1 iterations is 0.0608422
Training accuracy after 1 iterations is 98.3024%
gW0=0.9900 gW1=0.9900 gb0=0.9900
etaW0=0.5023 etaW1=0.5023 etab0=0.5023
|dW0|=0.01540 |dW1|=0.00740 |db0|=0.00188

Cross-entropy loss after 501 iterations is 0.029300808
Training accuracy after 501 iterations is 99.3214%
gW0=0.0905 gW1=0.0380 gb0=0.0087
etaW0=1.6529 etaW1=2.5316 etab0=5.0653
|dW0|=0.00714 |dW1|=0.00339 |db0|=0.00062

Final cross-entropy loss is 0.024607812
Final training accuracy is 99.4571%
CPU times: user 1h 18min 32s, sys: 1h 5min 8s, total: 2h 23min 41s
Wall time: 7min 48s
```

6 Predictions for testing data

[167]:
```
# Predictions
y_pred_test = np.argmax(h(X_test,W,b), axis=1)
```

[169]:
```
print(y_pred_test)
```

```
[2 0 9 ... 3 9 2]
```

[173]:
```
# Generating submission using pandas for grading
predictions = pd.DataFrame({'ImageId': range(1,len(X_test)+1) ,'Label': y_pred_test })
predictions.to_csv("simple_mnist_result.csv",index=False)
```

[176]:
```
predictions.head()
```

[176]:

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	9
4	5	3